A PROCESS-DRIVEN METHODOLOGY FOR MODELING SERVICE-ORIENTED COMPLEX INFORMATION SYSTEMS

Alfredo Cuzzocrea[1], Alessandra De Luca[2] and Salvatore Iiritano[2]
[1]*ICAR-CNR and University of Calabria, Calabria, Italy*
[2]*Exeura Srl, Rende, Italy*

Keywords: Service-Oriented Information Systems, Information Systems Design Methodologies.

Abstract: This paper extends state-of-the-art design methodologies for *classical* information systems by introducing an *innovative* methodology for designing *service-oriented information systems*. Service-oriented information systems can be viewed as information systems adhering to the novel *service-oriented paradigm*, to which a plethora of novel technologies, such as *Web Services*, *Grid Services* and *Cloud Computing*, currently marry. On the other hand, actual state-of-the-art literature encloses few papers that focus the attention on this yet-interesting research challenge. With the aim of fulfilling this gap, in this paper we provide a process-driven methodology for modeling service-oriented complex information systems, and we prove its effectiveness and reliability on a comprehensive case study represented by a real-life research project.

1 INTRODUCTION

While a lot of research has been done in the context of design methodologies for *classical* information systems (a significant excerpt of them is provided in Section 2), the issue of design effective and reliable methodologies for modeling *service-oriented information systems*, which, without loss of generality, can be viewed as information systems adhering to the novel *service-oriented paradigm* (Papazoglou & van den Heuvel, 2006; Papazoglou & van den Heuvel, 2007), is still a big research challenge. This is mainly due to the explosion of novel service-oriented technologies such as *Web Services*, *Grid Services*, *Cloud Computing*, and so forth.

Starting from this critical evidence, in this paper we propose an *innovative* methodology for modeling service-oriented information systems, which introduces several points of research innovation with respect to the state-of-the-art literature.

Our proposal falls in the context of *activity-based methodologies*, since it deeply leverages on the concept of *process*, and mostly focuses the attention on modeling *activities* to be performed within the scope of a given process, according to a *hierarchical abstract decomposition*. In more detail, our proposed methodology consists of the four following *hierarchical* design phases meaning that each phase f_i is used as a basis for the subsequent phase f_{i+1} in the terms that phase f_i produces in output a *formal model* that acts as input for phase f_{i+1}:

- *Analysis of Requirements*, which produces in output a BusinessModel model;
- *Conceptual Design*, which originates a ProjectModel model;
- *Logical Design*, which produces in output an ImplementationModel model;
- *Services Design*, which originates a ServiceModel model.

Interactions among the various phases of the proposed methodology follow a *feedback-waterfall methodology* (Royce, 1970) characterized by *incremental* and *iterative* procedures in which, at the end of each phase, all model instances originated by the previous phase are updated on the basis of the modeling of the actual phase.

The paper is organized as follows. In Section 2, we focus the attention on previous efforts which constitute the active literature for our research. Section 3 illustrates the *Analysis of Requirement* phase, and the BusinessModel model. In Section 4, the *Conceptual Design* phase is described, along with the ProjectModel model. Section 5 focuses the

Cuzzocrea A., De Luca A. and Iiritano S. (2010).
A PROCESS-DRIVEN METHODOLOGY FOR MODELING SERVICE-ORIENTED COMPLEX INFORMATION SYSTEMS.
In *Proceedings of the 12th International Conference on Enterprise Information Systems - Information Systems Analysis and Specification*, pages 390-398
DOI: 10.5220/0003020603900398

attention on the *Logical Design* phase, and also provides the description of the ImplementationModel model. In Section 6, the *Services Design* phase is illustrated, along with the ServiceModel model. Finally, in Section 7 we derive conclusions of our research, and draw directions for further efforts in this scientific field.

2 RELATED WORK

Two main literature contexts are relevant for our research. The first one concerns with design methodologies for classical information systems developed in the context of DBMS' first research experiences. The second one instead concerns with innovative methodologies for service-oriented information systems, which are more and more attracting the attention from a large community of researchers, mainly as a direct effect of novel service-oriented technologies (Papazoglou & van den Heuvel, 2006; Papazoglou & van den Heuvel, 2007) such as *Web Services*, *Grid Services*, *Cloud Computing*, and so forth.

We first focus the attention on design methodologies for classical information systems. The mutual relationship among business processes and information systems has been firstly studied in the early 90's by the pioneer paper (Davenport & Short, 1990), which puts in evidence how (*i*) business processes have a strong influence on both the final structure and functionalities of information systems, and, in turn, (*ii*) the design of specific business processes strongly depends on the internal organization of the target information system itself.

Since then, a plethora of process-based information system design methodologies have appeared in literature, and some interesting applications of them have been proposed as well. Among the most promising ones, we recall: (*i*) integration of process-oriented modeling methodologies and Data Warehouses (zur Muehlen, 2001), (*ii*) business processes simulation oriented to precisely capture information systems requirements (Serrano, 2003), (*iii*) process-driven modeling in the context of *e*-learning systems (Kim et al., 2005).

Based on this strong mutual interconnection between business processes and information systems, (Grover et al., 1994; van Meel et al., 1994) suggest that achieving a *total synergy* between design of business processes and development of information systems should be a goal for every business organization.

Nevertheless, (Earl, 1994) notices that, in real-life organizations, business analysts and information system engineers have very often distinct roles, make use of different tools, techniques and terminologies. Obviously, this *dichotomy* between business analysts and information system engineers makes the goal of integrating business processes and information systems far from being reached. On the other hand, (MacArthur et al., 1994) points out that it is very difficult to predict mutual consequences occurring in business organizations and information systems, and hence re-engineering becomes critical.

As regards relationships among available design approaches, (Giaglis, 2001) proposes a *taxonomy* of business processes and information systems modeling techniques, by also highlighting similarities and differences among state-of-the-art alternatives. Furthermore, (Giaglis, 2001) analyzes and systematizes the following perspectives that any information system should support: (*i*) *functional perspectives*, (*ii*) *behavioral perspectives*, (*iii*) *organizational perspectives*, and (*iv*) *informational perspectives*.

With respect to modeling languages, (Vasconcelos et al., 2001) presents an UML-based framework for modeling strategies, business processes and information systems, and proposes the adoption of a multi-level approach during the modeling phase. Likewise, (Castela et al., 2001; Neves et al., 2001) propose the usage of UML for capturing several aspects of information systems design. Following this trend, (Cuzzocrea et al., 2008) proposes a process-driven methodology for *continuous* information systems modeling, which makes use of *process mining techniques* (e.g., (Greco et al., 2005)) to improve the feedback design phases of the methodology.

As regards design methodologies for service-oriented information systems, few papers in the active literature investigate this yet-interesting research challenge. (Chung et al., 2007) first discusses principles of service-oriented information systems re-engineering, and proposes the integration of a classical business process engine for the execution of composite services together with pre-existing database applications in order to raise traditional legacy systems towards modern service-oriented information systems. (Arni-Bloch & Ralyté, 2008) focuses instead the attention on service-oriented information systems engineering, and proposes a *situation-driven* approach according to which an information system is viewed as a collection of service-shaped inter-related *method chunks*, and an innovative integration strategy is

proposed to achieve the comprehensive service-oriented information system. Finally, (Herold et al., 2008) studies the suitability of *Model-Driven Development* (MDD) paradigms to the issue of supporting the construction of service-oriented distributed enterprise information systems via directly deriving the design of software components from the underlying business processes of the target enterprise.

3 ANALYSIS OF REQUIREMENTS

Analysis of Requirements is characterized by three main (sub-)activities: (*i*) identification of *actors*, i.e. the entities of the external world which interact with the information system; (*ii*) modeling of the *information* managed by the organization in form of *archives* (i.e., data/information sources); (*iii*) modeling of *processes* of the target organization to be captured and implemented by the information system.

Data, information and knowledge collected during the *Analysis of Requirements* phase (e.g., by means of interviews) are formally modeled by a BusinessModel model. Each BusinessModel model consists of the following three models (see Figure 1): (*i*) BusinessActorSchema model, (*ii*) ArchiveSchema model, and (*iii*) ProcessSchema models. These models are then instantiated as three corresponding *schemas* that aim at representing, according to logically-separated areas, concepts characterizing the *initial* design phase of the information system, with respect to actors, archives and processes, respectively. In more detail, a BusinessActorSchema model represents actors of the system along with their *hierarchical relations* (e.g., *Manager* ← *Employee*). An ArchiveSchema model contains archives representing data and information sources of the information system (e.g., *Invoices*, *Sales*). A ProcessSchema model represents information related to the processes of the information system (e.g., *Invoicing*, *Hiring*). In particular, a ProcessSchema model is exploited to describe *interdependence relations* among processes, thus modeling the *value chain* of the enterprise being modeled.

During the design of processes, the natural decomposition of processes into *sub-processes* and, recursively, *activities* must be mandatorily taken into account, as well as for the associations among processes/sub-process/activities and the involved

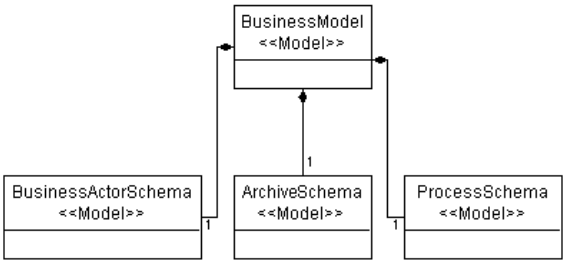

Figure 1: Meta-Model of the model BusinessModel.

actors and archives. Both process hierarchical organization and associations with actors/archives are modeled by the meta-model of the model Process depicted in Figure 2.

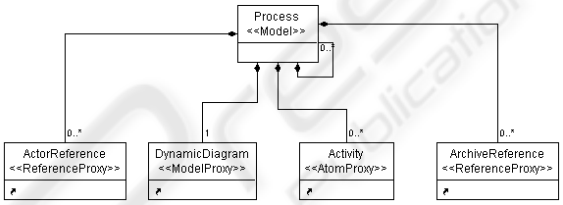

Figure 2: Meta-Model of the model Process.

In particular, as illustrated in Figure 2, a Process model represents the *static structure* of a process, consisting of the following components: (*i*) processes and sub-processes (a sub-process is a process itself), modeled by the Process model, which represent main functionalities/procedures of the target organization; (*ii*) atomic activities, modeled by the Activity class, which are atomic entities describing *elementary operations* which are conceptually no further decomposable into simpler operations (i.e., activities do not have a proper structure); (*iii*) associations to actors, modeled by the ActorReference class pointing to the BusinessActorSchema model previously-defined; (*iv*) associations to archives, modeled by the ArchiveReference class pointing to the ArchiveSchema model previously-defined; (*v*) a *dynamic diagram*, captured by the DynamicDiagram model, which enable us to model dynamic aspects of processes, thus the *activity flow* of the information system, along with *pre-conditions* and *post-conditions* useful to connote and make richer the overall *dynamicity* of the information system.

In particular, a DynamicDiagram model allows us to model dynamic aspects of those processes composed by multiple activities. Therefore, activities, modeled by the Activity class, are also basic components of dynamic diagrams, like for

processes, but with the difference that a DynamicDiagram model focuses on dynamic aspects of the information system, whereas a Process model focuses on static aspects of the information system. In a DynamicDiagram model, each Activity class is labeled by a label indicating the *action* to be performed by the activity itself. Furthermore, an Activity class can be labeled as *Start Activity* (respectively, *Final Activity*), representing a starting (respectively, final) activity of a process. A DynamicDiagram model may also include so-called *conditional branches*, which are instances of the class Branch. Conditional branches allow us to specify alternative flows of execution of the process, which are activated based on the Boolean value of pre-defined conditions. Conditional branches are graphically represented by a diamond having a single input transition, denoted by T^{in}, and two or more output transitions, denoted by T_i^{out}, such that $i \in \{0, 1, \ldots, n\}$. Each output transition T_i^{out} is associated with a *branching condition*, denoted by $Cond(T_i^{out})$. Branching conditions are required to be *mutually exclusive*, which imposes that only one condition at a time can be satisfied. More formally, output transitions T_i^{out} are required to satisfy both the following equalities: (*i*) $\bigvee_i Cond\ (T_i^{out}) = 1$ and (*ii*) $\bigoplus_i Cond\ (T_i^{out}) = 1$, such that symbols $\vee$ and $\oplus$ denote the logical OR and XOR operators, respectively. Semantics associated with the branch component is as follows. Conditions associated to output transitions are evaluated upon the activation of the input transition, and, among all the possible (*n*) ones, only the *singleton* output transition having the Boolean condition equal to TRUE is activated. Likewise, it is possible to compose multiple transitions to capture more complex (logical) conditions by means of a Merge class. A DynamicDiagram model may also include parallel executions of transitions. This of course requires somewhat synchronization. To this end, we make use of the construct Fork and Join. Fork has a singleton input transition and two or more output transitions. Fork's semantics is as follows: as soon as the input transition is activated, *all* the output transitions are started *in parallel*. Conversely, Join has two or more input transitions and one (singleton) output transition, which is activated when *all* the input transitions are activated.

4 CONCEPTUAL DESIGN

As highlighted in Section 1, the *Conceptual Design* phase produces in output the ProjectModel model. In turn, a ProjectModel model consists of the following four models (see Figure 3): (*i*) ProjectActorSchema model, (*ii*) DataSchema model, (*iii*) ViewSchema model, and (*iv*) FunctionSchema model. Similarly to the hierarchical organization of the BusinessModel model (see Section 3), each of these models are then instantiated as four corresponding *schemas* that aim at representing, according to logically-separated areas, concepts characterizing the *second* design phase of the information system.

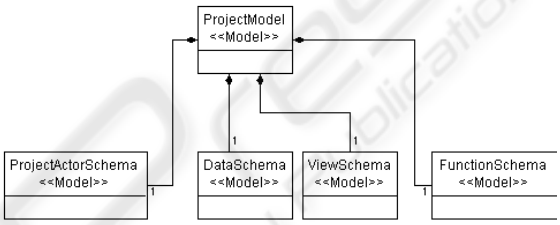

Figure 3: Meta-Model of the model ProjectModel.

Let us now focus on each of the (sub-)models of the ProjectModel model. A ProjectActorSchema model represents so-called *active actors* of the information system, i.e. those actors that interact with the information system effectively, along with their hierarchical relations. A DataSchema model captures conceptual representations of data sources handled by the information system. These conceptual representations are similar to well-known ER models from DBMS technology. In every conceptual representation of data sources further levels of abstraction are necessary. This in order to cope with the different views over the data sources themselves used by different functionalities/procedures of the information system. These views are captured by the ViewSchema model, which comprises a set of View classes, each one being a *projection* over the whole data source targeted to support a specific functionality/procedure of the information system. A FunctionSchema model describes functions and dependencies among functions by means of an approach similar to the one used to model processes (see Section 3). The meta-model of the model FunctionSchema is shown in Figure 4.

A FunctionSchema model comprises the components Function and FunctionDependency. Function is a model that describes information

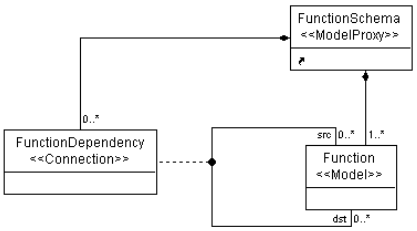

Figure 4: Meta-Model of the model FunctionSchema.

system functionalities/procedures by means of a hierarchical composition of *sub-functions* and *elementary functions*, just like processes are organized into sub-processes and atomic activities (see Section 3). FunctionDependency is a class that models dependencies among sub-functions and elementary functions, respectively, such as pre-conditions to alternative executions. For the sake of explanation, a FunctionDependency class is able to formally express conditions in the form of: "*before function F_i is executed, function F_j must have completed*".

Analogously to what happens with processes (see Section 3), the static description of a function is modeled by the meta-model of the model Function depicted in Figure 5, which allows us to capture the associations of a function with components needed during its execution.

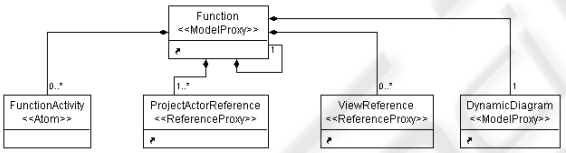

Figure 5: Meta-Model of the model Function.

In particular, as illustrated in Figure 5, a Function model consists of the following components: (*i*) functions and sub-functions (a sub-function is a function itself), modeled by the Function model, which represent functionalities/procedures of the information system; (*ii*) elementary functions, modeled by the FunctionActivity class, which are atomic entities describing elementary procedures implemented by the information system (e.g., accessing a database); (*iii*) associations to actors, modeled by the ProjectActorReference class pointing to the ProjectActorSchema model previously-defined; (*iv*) associations to views, modeled by the ViewReference class pointing to the ViewSchema model previously-defined; (*v*) a dynamic diagram, captured by the DynamcDiagram model, which enables us to model the dynamic behavior of functions, similarly to what happens in modeling dynamic aspects of processes (see Section 3).

Constructing a ProjectModel model is an *incremental* and *iterative* task that comprises several well-separated steps. The first step consists of a *raw* modeling of views. In the second step, the global ProjectModel model is sketched, based on views of the previous step. On the basis of the global ProjectModel model so far obtained, the first step is re-executed iteratively until a *refined* modeling of views is achieved. At this point, a refined definition of the global ProjectModel model can be obtained based on the refined definition of views, and so forth, in a feedback-like manner. This (sub-)task is iterated until a *sufficient degree of detail* in the definition of the global ProjectModel model is achieved. It is worth to remark that the task of modeling a ProjectModel model is *intrinsically non-deterministic*, and a *gap* between the BusinessModel model and the ProjectModel model exists. This gap must be filled by means of *best* modeling practices, project experiences, technological know-how and engineering methodologies.

5 LOGICAL DESIGN

As illustrated in Section 1, the *Logical Design* phase of our proposed methodology produces in output the ImplementationModel model, which consists of the following five models (see Figure 6): (*i*) RelationalSchema model, (*ii*) ControlSchema model, (*iii*) InterfaceSchema model, (*iv*) ComponentSchema model, and (*v*) ArchitectureSchema model.

Similarly to what happens with BusinessModel and ProjectModel models, each model in the ImplementationModel model is then instantiated by means of a corresponding schema which aims at representing, according to logically-separated areas, concepts characterizing the *third* design phase of the information system.

In the remaining part of this Section, we provide an in-depth explanation of each model characterizing the meta-model of the ImplementationModel model. A RelationalSchema model contains elements necessary to describe the structure of the relational database underlying the information system being modeled. Such a model is designed on the basis of the DataSchema model defined in the

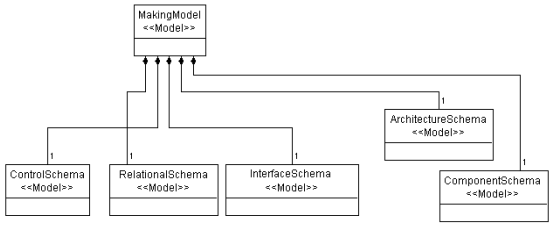

Figure 6: Meta-Model of the model ImplementationModel.

ProjectModel model. In this modeling phase, for each entity previously-defined in the DataSchema model a relational table with the same name is created in the RelationalSchema model Tables created in this phase are characterized by the presence of some *additional* and *more-detailed* information than the corresponding entities defined in the DataSchema model, such as *type* and *range of values* of attributes, *referential constraints*, and so forth. In addition to this, relations among tables and constraints over such relations, like *cardinality* and *foreign-key* constraints, are modeled as well.

A ControlSchema model is used to design a *middleware layer* between the user interface layer and the database layer, respectively, in order to achieve a greater degree of independence among (software) components of the information system being modeled. A ControlSchema model describes functionalities/procedures of the information system that perform read/write operations on the data sources defined in the RelationalSchema model. Each function described in the ControlSchema model is also linked to the table on which it executes in the RelationalSchema model, by means of a *dependence relation*. When functions operate on multiple tables rather than only one, they may be linked to a View class from the ViewSchema model of the ProjectModel model.

An InterfaceSchema model defines the structural requirements of the interface used by the information system, and models the interactions occurring between the interface layer and the control layer, respectively. *Forms*, captured by the Form class, are the main components of this modeling phase. In our proposed methodology, a Form class is modeled as an *aggregation* of Unit classes. A Unit class represents a logical/physical portion of the Form and can be one of the following (specialized) classes: (*i*) DisplayUnit, (*ii*) EntryUnit, and (*iii*) DataUnit class, respectively. A DisplayUnit class is intended to model *static data*, i.e. data that are not retrieved by the underlying relational database, but, instead, is simply inserted into the Form class as *static unmodifiable field*, i.e. titles of menus, descriptions of fields, and so forth. On the other hand, EntryUnit instances capture *input fields* by means of which users submit data, such as parametric fields, query-aware dates, and so forth. A DataUnit class models data extracted from the underlying relational database that must be displayed to the user. Typically, a DataUnit instance is displayed on behalf of users throughout queries submitted by means of EntryUnit instances.

In order to enrich the expressive power of the InterfaceSchema model, Form instances can be linked one another by means of the component Link, with the goal of modeling possible interaction scenarios. Instances of the Link class can be one of the following specialized (sub-)classes: (*i*) SimpleLink, or (*ii*) ParamLink class, respectively. In more detail, a SimpleLink represents an *oriented link* between two instances of Form class, whereas a ParamLink models an *oriented link* where somewhat information exchange between source Form and destination Form needs to be performed. As an example, the use of ParamLink makes it possible to model the interaction scenario in which a user submits a query to the information system by means of the source Form instance, and then he/she visualizes the query answer by means of the destination Form instance.

Furthermore, in order to achieve a *much modular* and *cleaner* representation, it is advisable to group together all Form instances related to the same logical area (or sub-area) into an Area model. Each Area model can contain further Area instances along with Form instances and other elements from the ControlSchema model previously-defined, and belonging to the same functional area. When modeling Area instances, with a little abuse of notation, a Link instance can even occur between a Form instance and an Area instance as well.

A ComponentSchema model describes a set of software components along with their mutual interdependency relations, thus giving a *high level view* of the entire information system. Such an abstraction allows the designer to: (*i*) model the different layers of the information system; (*ii*) group together several control elements previously-defined; (*iii*) represent software objects located of the information system, such as executable programs, libraries, files, and so forth.

Finally, an ArchitectureSchema model is exploited to model the hardware/software architecture of the information system to be deployed. In particular, a network-based architecture is very-often advocated, so that a set of *architecture nodes*, captured by the

class Node, are determined, and ad-hoc interconnections/protocols among them are derived accordingly. A Node class models an *abstract computational unit*, usually being a hardware device. In turn, each Node class may contain a number of atomic elements captured by the class Component, usually being software modules modeled in the ControlSchema model.

6 SERVICES DESIGN

The fourth phase of our proposed methodology is the *Services Design* phase, which produces in output a ServiceModel model (see Section 1). This phase is introduced in order to address the following two issues: (*i*) make it possible to model procedures/functionalities exposed by the information system by exploiting the service-oriented paradigm; at the same time, (*ii*) enable the development of service-oriented applications without affecting the models developed by means of the three previous design phases of the proposed methodology.

In particular, the aim of our proposal is to devise a *flexible* information systems design methodology capable of easily supporting the conversion of a traditional information system into a service-oriented one, thus adding a "service-oriented flavor" to *legacy* information systems, and enabling an easy transition from a traditional *three-phase* design methodology to a *four-phase* one accordingly, where the final product is represented by the ServiceModel model.

The ServiceModel model produced as output by the *Services Design* phase is structured on a hierarchy comprising the following four models (see Figure 7): (*i*) ServiceSchema model, which describes functionalities exposed by the information system in forms of services deployed in the context of a service-oriented architecture; (*ii*) ServiceDataSchema model, which represents data sources on top of which the previous service-oriented procedures/functionalities execute; (*iii*) ServiceControlSchema model, which captures a sub-set of hidden-to-the-user service-oriented functions necessary to support the procedures/functionalities exposed by the information system; (*iv*) ServiceInterfaceSchema model, which models user interfaces and their interactions with service-oriented procedures/functionalities defined in the ServiceSchema model.

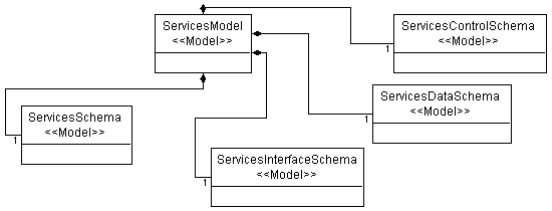

Figure 7: Meta-Model of the model ServiceModel.

In more detail, the ServiceDataSchema model allows us to model data sources on which service-oriented procedures/functionalities execute. As illustrated in Figure 8, data sources modeled by the ServiceDataSchema model fall into the following broad categories. The first category comprises references to relational tables modeled in the RelationalSchema model of the ImplementationModel model. This kind of data is captured by means of the class EntityReference. On the other hand, data belonging to the second category models information/metadata necessary to the proper service management and control, and are modeled within the ServiceDataSchema model by means of the class ServiceEntity. The class ServiceEntityConnection is instead exploited to create and manage logical references between services and relational tables on top of which services execute.

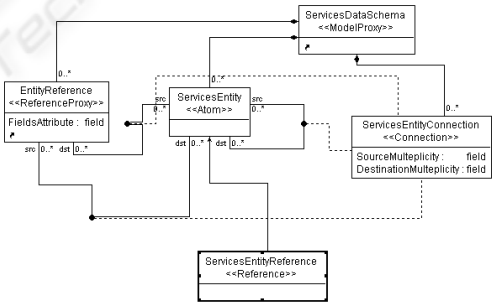

Figure 8: Meta-Model of the model ServiceDataSchema.

The ServiceControlSchema model (see Figure 9) is introduced to model baseline services necessary to support the same service-oriented paradigm. This component of the proposed methodology makes use of well-known reference-based service deployment and orchestration paradigms.

The ServiceSchema model allows us to model procedures/functionalities exposed by the information system in a service-oriented manner. As shown in Figure 10, the ServiceSchema model is composed by the following three models, each of

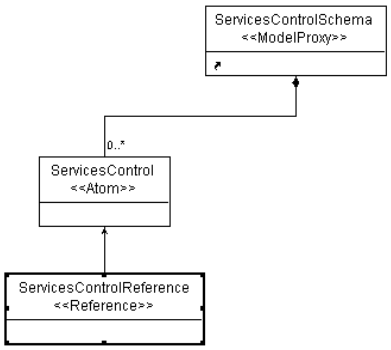

Figure 9: Meta-Model of the model ServiceControlSchema.

them identifying a specific *class of services*: (*i*) WrappingWebServices model, which models in a service-oriented manner procedures/functionalities previously defined as control components within the ControlSchema model of the ImplementationModel model; (*ii*) SupportWebServices model, which models services supporting the service-oriented architecture itself; (*iii*) WorkflowWebServices model, which enables us to model orchestration and coordination primitives for particular kind of services such as *wrapper* services and *support* services, based on the formalism and the wide availability of constructs of *workflows* (e.g., (Greco et al., 2005)).

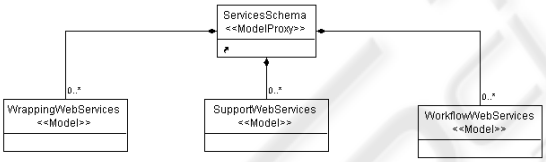

Figure 10: Meta-Model of the model ServiceSchema.

Let us focus in great detail on all these kinds of services captured by the design methodology we propose, and modeled within the ServiceSchema model. As mentioned before, *Wrapping Web Services* represent the service-oriented implementation of procedures/functionalities exposed by the information system, via directly looking at control components of the ControlSchema model (which, in turn, is contained by the ImplementationModel model). This particular deployment mechanism imposes us to define ad-hoc wrapping components with suitable *state-parameters* and *methods*, beyond to implement the corresponding control components in the ImplementationModel model devoted to effectively make the middleware between procedures/functionalities of the information system and its service-oriented realizations. *Support Web Services* are needed to support the service-oriented paradigm at run time. Similarly to the case of *Wrapping Web Services*, designing *Support Web Services* involves in designing ad-hoc support (software) objects within the software infrastructure, beyond to the corresponding control components within the ImplementationModel model. Finally, *Workflow Web Services* are the most critical services in our proposed design methodology as they deal with the issue of providing orchestration and coordination primitives to both *Wrapping Web Services* and *Support Web Services*, respectively. The complete meta-model of the WorkflowWebServices model is shown in Figure 11. Due to its inherent complexity and for space reasons, we only provide a description of its (interior) model that plays the major role, i.e. the WorkflowDiagram model.

The WorkflowDiagram model focuses on the modeling of orchestration and coordination primitives over services implemented within the information systems via well-consolidated workflow formalisms. From this evidence, a clear hegemony of the WorkflowDiagram model follows.

Finally, coming back to the description of components of the ServiceModel model, the ServiceInterfaceSchema model allows us to capture and describe ad-hoc interfaces that are in charge of supporting interactions between users and services. In particular, this is achieved via the design of suitable forms that directly build on the workflows defined in the WorkflowDiagram model.

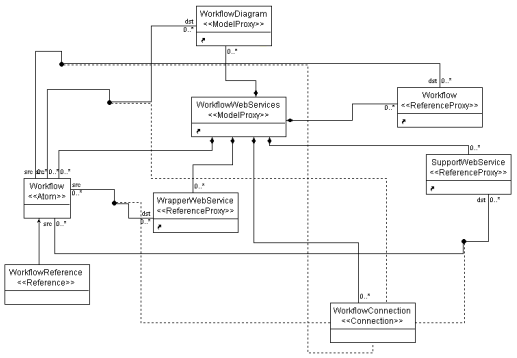

Figure 11: Meta-Model of the model WorkflowWebServices.

7 CONCLUSIONS AND FUTURE WORK

Starting from actual limitations of state-of-the-art

service-oriented information systems design methodologies in capturing both complexity and new requirements dictated by the emerging service-oriented paradigm, in this paper we have introduced an innovative methodology for modeling service-oriented information systems, which embeds several points of research innovation with respect to the active literature.

The essence of our proposal relies in a process-driven modeling of the information system functionalities/procedures, which are then enclosed in ad-hoc routines exposed as services by the reference architecture on top of which the target information system is deployed. This strategy has already demonstrated its effectiveness and reliability in a number of real-life realizations of complex service-oriented information systems.

Future work of our research is actually oriented towards two different goals: (*i*) devising a complete suite able to support all the design phases of service-oriented information systems by also including additional features such as *monitoring* and *continuous re-engineering* of the at-work information system; (*ii*) adding novel characteristics to our methodology, such as the amenity of *automatically* generating wrapper (software) components for functionalities/procedures of the information system from the business modeling layer directly, and the amenity of embedding *active behaviors* (like in the style of well-known ECA rules of DBMS) across all the modeling phases of the methodology.

REFERENCES

Arni-Bloch, N., and Ralyté, J., 2008. Service-Oriented Information Systems Engineering: A Situation-Driven Approach for Service Integration. In *Proc. of the 20th CAiSE Int. Conf.*, pp. 140-143.

Castela, N., Tribolet, J.M., Silva, A., and Guerra, A., 2001. Business Process Modeling with UML. In *Proc. of the 3rd ICEIS Int. Conf.*, Vol. 2, pp. 679-685.

Chung, S., Byung Chul An, J., and Davalos, S., 2007. Service-Oriented Software Reengineering: SoSR. In *Proc. of 40th IEEE HICSS Int. Conf.*, pp. 172-181.

Cuzzocrea, A., Gualtieri, A., and Saccà, D., 2008. *A Process-Driven Methodology for Continuous Information Systems Modeling*. In *Proc. of the 10th ICEIS Int. Conf.*, Vol. 2, pp. 82-88.

Davenport, T. H., and Short, J. E., 1990. The New Industrial Engineering: Information Technology and Business Process Redesign. In *Sloan Management Review*, Vol. 31, No. 4, pp. 11-27.

Earl, M. J., 1994. The New and the Old of Business Process Redesign. In *Journal of Strategic Information Systems*, Vol. 3, No. 1, pp. 5-22.

Giaglis, G.M., 2001. A Taxonomy of Business Process Modeling and Information Systems Modeling Techniques. In *International Journal of Flexible Manufacturing Systems*, Vol. 13, No. 2, pp. 209-228.

Greco, G., Guzzo, A., Manco, G., and Saccà, D., 2005. Mining and Reasoning on Workflows. In *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 4, pp. 519-534.

Grover, V., Fielder, K.D., and Teng, J.T.C., 1994. Exploring the Success of Information Technology Enabled Business Process Reengineering. In *IEEE Transactions on Engineering Management*, Vol. 41, No. 3, pp. 276-284.

Herold, S., Rausch, A., Bösl, A., Ebell, J., Linsmeier, C., and Peters, D., 2008. A Seamless Modeling Approach for Service-Oriented Information Systems. In *Proc. of 5th IEEE ITNG Int. Conf.*, pp. 438-446.

Kim, K-H, Yoo, H.-J., and Kim, H.-S., 2005. A Process-Driven E-Learning Content Organization Model. In *Proc. of 4th IEEE ACIS Int. Conf.*, pp. 328-333.

MacArthur, P.J., Crosslin, R.L, and Warren, J.R., 1994. A Strategy for Evaluating Alternative Information System Designs for Business Process Reengineering. In *International Journal of Information Management*, Vol. 14, No. 4, pp. 237-251.

van Meel, J.W., Bots, P.W.G., and Sol, H.G., 1994. Towards a Research Framework for Business Engineering. In *IFIP Transactions A: Computer Science and Technology*, Vol. 54, pp. 581-592.

zur Muehlen, M., 2001. Process-Driven Management Information Systems - Combining Data Warehouses and Workflow Technology. In *Proc. of the 4th ICECR-4 Int. Conf.*, pp. 550-566.

Neves, J., Vasconcelos, A., Caetano, A., Sinogas, P., Mendes, R., and Tribolet, J.M., 2001. Unified Resource Modelling: Integrating Knowledge into Business Processes. In *Proc. of the 3rd ICEIS Int. Conf.*, Vol. 2, pp. 898-904.

Papazoglou, M.P., and van den Heuvel, W.-J., 2006. Service-Oriented Design and Development Methodology. In *International Journal of Web Engineering and Technology*, Vol. 2, No. 4, pp. 412-442.

Papazoglou, M.P., and van den Heuvel, W.-J., 2007. Service Oriented Architectures: Approaches, Technologies and Research Issues. In *VLDB Journal*, Vol. 16, No. 3, pp. 389-415.

Royce, W.W., 1970. Managing the Development of Large Software Systems. In *Proc. of the 1970 IEEE WESCON Int. Conf.*, pp. 1-9.

Serrano, A. 2003. Capturing Information System's Requirement Using Business Process Simulation. In *Proc. of the 15th ESS Int. Conf.*

Vasconcelos, A., Caetano, A., Neves, J., Sinogas, P., Mendes, R., and Tribolet, J. M., 2001. A Framework for Modeling Strategy, Business Processes and Information Systems. In *Proc. of the 5th IEEE EDOC Int. Conf.*, pp. 69-80.